SPAcENoCs : A SCALABLE PLATFORM FOR FPGA ACCELERATED EMULATOR OF NoCs

A Thesis

by

GUANGMING CHEN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Paul V. Gratz
Committee Members,	Jiang Hu
	Eun Jung Kim
Head of Department,	Chanan Singh

May 2013

Major Subject: Computer Engineering

ABSTRACT

The majority of modern high performance computing systems have employed on-chip multi-processors. As the number of on-chip cores soars, the traditional non-scalable communication infrastructures, commonly observed as shared buses or cross-bars, no longer accommodate the increasing communication demand by the modern multi-core chips. The newly emerging Network-On-Chip (NoC) interconnection scheme has provided a scalable, robust and power-efficient solution that also satisfies the requirements on both bandwidth and latency. A tool that enables swift exploration of the vast NoC design space is then in great demand to meet the stiff time pressure over research and development.

Based on the work of AcENoCs, an NoC simulator designed on the basis of software and hardware codesign seeking for a large simulatable network size, the SPAcENoCs (**S**calable **P**latform for FPGA **Ac**celerated **E**mulator of **NoCs**) employs the Time-Division Multiplexing (TDM) techniques to implement a simulator for even larger NoCs without sacrificing simulation speed and cycle accuracy which have been highlighted in the work of AcENoCs. This paper will focus on re-organization of the given software/hardware codesigned frameworks so that the TDM techniques may be applied. While both frameworks require re-design, the major efforts involve re-construction of the hardware framework by adding data buffers and affiliated logic to ensure the data generated in different time divisions are properly preserved and transmitted. Various design tradeoffs over hardware budget and simulation performance are also discussed and attempted in this paper. During the development process, the techniques of device virtualization and generic programming are introduced to overcome the verification challenges that are commonly seen in software/hardware

codesigned systems.

The synthesis results of various design options suggested that the simulation of a 9×6 network, more than twice the size of largest applicable size in AcENoCs, can be accommodated by the device. Based on the simulation result of AcENoCs, the estimated speedup of SPAcENoCs over software simulator for the 9×6 NoC is around 28-94X, twice the one achieved by AcENoCs in a smaller network.

To my Family and Friends for their love and encouragement

ACKNOWLEDGMENTS

It would never be sufficient for me to express my gratitude towards the people without whose support and help this work would ever be complete. I am highly indebted to my advisor Dr. Paul V. Gratz for his wisdom, patience, guidance and encouragement that have both materially and spritally supported me during the years of my graduate study. His excellent expertise and vision in teaching and research guided me to my current career road that may continue over time. I would also like to appreciate my committee members, Dr. Jiang Hu and Dr. Eun Jung Kim, for their valuable and timely advices throughout my academic life in Texas A&M. Also, I want to thank all the supporting faculty and staffs of ECE Department and Texas A&M University for setting up a nice academic and campus environment.

I also take this opportunity to thank Ms. Xue Yang, my research partner on the SPAcENoCs project, for all her efforts that accelerated the progress of the research. I would also like to thank all CAMSIN research group members, the ones I have or have not seen, for the inspiring and warming research atmosphere that they have been forming through their constant efforts.

No words are enough to express my gratitude to my family members for their unconditional love, support and encouragement. My wife Xiaomeng, my parents Peng and Weiqing, and my grandparents have sacrificed quite a lot and provided me the best possible love and support. Whatever I am today and will be tomorrow is because of them.

Finally, special thanks to all my friends, regardless of where they live, for their time spent with me for my happy or sad moments.

TABLE OF CONTENTS

LIST OF TABLES

TABLE Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

As the size of transistors has been shrinking in computer processor manufacture industry, people has steered to multi-core processor scheme to facilitate high performance computing from developing single heavyweight superscalar processor whose performance-to-power ratio and Instruction Level Parallelism (ILP) have been limited by the serialized part of applications as describe by Amdahl's law[1, 2]. Under a multi-core scheme, multiple processors, scalar or superscalar, with certain communication connections for data sharing and synchronization are instantiated on a single silicon chip. Each single processor or processing unit usually has narrower dispatch width compared to a heavyweight superscalar one, but presents better performance efficiency by avoiding area and power overhead required to enable a heavyweight superscalar processor.

A new issue has then emerged and still remains as a key design challenge on how inter-processor communication should be implemented to insure high throughput and low latency data sharing and exchanges. The bus based communication network, where a few senders and receivers share one set of bus, are used in systems with limited number of processors, or nodes, to achieve high efficiency data exchange. However, this traditional method does not scale well if it is deployed in a cluster with increased number of processors. While multiple nodes keep requesting the privilege of using the bus with iterations of request-conflict-request attempts, the bus is effectively kept idle. As the total number of nodes and concurrent bus requests grows, the probability of conflicts increases, wasting a significant portion of bus throughput and power. In a system where infinite nodes communicate over one shared bus, the

effective throughput is less than 40 percent of the bandwidth of the bus[3].

A new communication scheme, the Network-on-Chip (NoC), also called on-chip interconnection network, designed by Dally and Towles, has been a popular scheme for multi-core high performance computing architecture for the very sake of a robust, high throughput, low latency and power efficient communication[4, 5]. The concept of NoC has been applied in several products by commercial companies and research institutions[6, 7, 8]. In an NoC architecture, multiple cores or function units are attached to routers interconnected by router-to-router, or node-to-node, links. The data requests and responses are transferred through these links between the nodes. A data packet is divided into smaller slices, called flits, to fit into width of inter-node connections, and traverses through the connections from sender (source) to the receiver (destination). If the source and destination nodes are not directly connected, the flits are then routed through several intermediate nodes. As there are multiple routes between any source-to-destination pair, the nodes may communicate with their peers concurrently without intervening and intervened by other peers, and thus the congestions on a bus-based system may be greatly relieved. Besides, as alternative routes exist between any pair of nodes, the system, with carefully designed data routing algorithm, may be able to recover from occasional runtime link faults, which is very difficult for a system based on shared bus.

Several design focuses factor into the ultimate performance and power consumption of a proposed NoC. The topology of the network determines the floor plan of the processing nodes and their attached routers and the way how these nodes are connected. The micro architecture of the router has a great impact on the latency that one data flit will face when it is traversed from a router's input port to the output port, determining the latency and throughput of the overall network. As the congestion rate of an NoC depends on the routing algorithm, the latter one has also

become an popular focus that interests most NoC researchers.

As all these factors interact and form a vast design space, there has been a great amount of researches seeking the optimal selection of design decisions for different application scenarios. However, it is usually difficult to explore the design space in a short period of time, especially in an era when time-to-market requirement has become a key pressure for a successful product. Thus, powerful tools that enable quick exploration of the design space have been studied a lot and cannot be emphasized any more.

A. Thesis Objective

Simulators for on-chip networks, supporting emulations with various detail levels and collection of crucial factors like performance and power dissipation, are employed for analysis and evaluation of a proposed on-chip network. A C/C++/SystemC based simulator usually emulates behavioral functionality of routers in an on-chip network. HDL simulators are employed for bit- and cycle- accurate emulations by running Register Transit Level (RTL) model written in Hardware Description Language (HDL) like VHDL or Verilog HDL.

Various levels of accuracy and details these emulators may provide, they are all run sequentially due to very limited parallel computing capability of both programming language and PC platform. As such NoC simulations, focusing on updating internal states of multiple routers that work concurrently, cannot utilize the parallelism inherited in an interconnection network. It even takes a long period of time for such a simulator to complete the emulation over a small or moderate scaled on-chip network. Besides the limitation of parallelism, it usually takes hundreds to thousands of processor cycles to update a single component's state update in one cycle, making

the simulation even slower.

Real hardware devices are then employed for acceleration of NoC simulations as they can execute actual data operation in place of cycle-consuming emulation of data flows. Also, one can instantiate multiple identical or similar hardware modules to further accelerate the speed of simulation by avoiding sequential emulation. FPGA is one of the most popular hardware accelerator solutions. Besides the advantages mentioned above, FPGA devices are designed with reconfigurability, and may be quickly deployed in respond to frequent changes of requirements during the evaluation of interconnection networks with various configurations. In real practice, a single-chip FPGA can accommodate an on-chip network whose size varies from 2×3 to 5×5, depending on detail level of implementation and hardware resources of the chip. Such schemes, with an FPGA running at 100-200 MHz, present one or two magnitude of acceleration over a PC platform running at 2-3 GHz in terms of simulation time.

However, fast and flexible as is an FPGA-based emulator, it is restricted to simulations for small and moderate scale interconnection network due to its logic resource constraints. Also, the simulation coordinator consists of complex control logic which is difficult to implement and parallelize. Consequently, an FPGA-accelerated simulator is usually divided into software and hardware partitions for a tradeoff between performance, complexity and efficiency of the design flow, and simulated scale. AcENoCs is one of the FPGA-based NoC simulation accelerators, and its largest applicable simulation scale is 5×5 due to the mentioned limitation.

This paper, the SPAcENoCs, presented an attempt, based on the work of AcENoCs[9, 10, 11], which furthers the exploration of the design space by employing TDM techniques. A large scaled NoC network, typically larger than 4x4, can be divided into such slices that each of them may fit into the single FPGA chip. If properly coordinated, these slices can be simulated one each time using the same hardware without

intervention with others. The division plan is flexible depending on the hardware resource budget and size of the entire network. This paper presented the implementation of SPAcENoCs with compile-time reconfigurability that supports variable sizes of both entire network and divided slices. By employing TDM techniques, the research also demonstrates the possibility of emulating large scale NoC, larger than the scale that one single FPGA chip may normally accommodate.

B. Thesis Organization

The remainder of the paper will be organized as follows. Chapter II discusses previous related works that utilized different trading-off strategies for various design prospects. Chapter III reviews the design of AcENoCs on the basis of which SPAcENoCs is implemented, and presents the SPAcENoCs' implementation of software and hardware frameworks that are beyond the design of AcENoCs. Chapter IV demonstrates the techniques to overcome the verification challenges and evaluates performance of the implementation. At last Chapter V concludes the work of SPAcENoCs and proposes several design prospects as future work.

CHAPTER II

RELATED WORK

Because of the need for exploration of the vast design space formed by various NoCs design factors and for locating critical design and performance bottlenecks into an early state of NoCs' design flow, several types of NoC simulation frameworks and tools have been proposed with different design emphasis and highlights. These methodologies have focused on simulation speed for reduced validation time, or simulation details for more evaluation accuracy, or both. Regardless of diversified simulation languages and models these works have used, the methodology can be generally characterized into two classes – the software and FPGA accelerated emulators.

A. NoC Software Simulators

Software NoC simulators may perform all levels of abstraction based on the tradeoffs between the focuses of simulation speed and simulation accuracy. Generally, as a simulation with lower level of abstraction (with more details and accuracy) requires more states evaluation and updates, the emulation time increases. Thus, for the sake of tolerable simulation time, some of the software simulators, usually using System C and its extensions as the barebones, performed system level simulation with limited details. Kogel et al.[12], Lu et al.[13], and Pestana et. al[14] all proposed System C based simulators providing ability for general comprehension on impacts of high level design factors over the NoCs' performance. All these simulators traded simulation accuracy off, due to the nature of System C, for prompt evaluation and validation of proposed NoCs.

On the other hand, the study on network topology requires cycle accurate sim-

ulators for interested researchers to aid their design and evaluation flow at the cost of simulation speed. The Proteo NoC[15] and OCIN_TSIM[16], written in System C or C++, with reconfigurability, cycle accuracy and support for diversified topologies, were all considered slow in terms of simulation speed. Some other cycle-accuracy seekers tried to reduce the simulation time by combining languages with different speed and accuracy characteristics. The modules that require higher level of abstraction can be run on a less-accurate-but-faster language to increase the simulation speed, while the modules needing high accuracy can be simulated by a more accurate language that takes longer to finish. The work of Goossens et al.[17] was one of the examples, which combined System C for transaction level simulation, and VHDL for cycle accurate emulation. However, the authors had to carefully handle the mismatch of accuracy between the two sides and special additional were needed to synchronize the data exchange caused by different simulation speeds.

Hard as all these works had tried to manage both level of details and speed of simulation without compromising each other, the researchers could never circumvent the sequential nature of any software simulator. As the detail level of simulation furthers and the size of the simulated network scales, the simulation time increases accordingly as it scales proportionally to the detail level and simulation size. As a result, the speed and accuracy of the simulation become constraining factor on each other and tradeoffs have to be made.

B. FPGA Based NoC Emulators

FPGA based emulation schemes have then come into the sight of NoC researchers as the accuracy-speed dilemma can be mitigated by the devices' parallel nature. Genko et al.[18, 19] proposed an FPGA based scheme on Xilinx Vertex II Pro device con-

taining an embedded PowerPC processor. In their work, the traffic generator, traffic receptor, and the simulated network were all implemented with hardware resources commonly known as look-up tables and flip flops. The traffic generators supported both stochastic and realistic traffic types, and the associated source queue were statically decided. The Xpipes compiler[20] was utilized to generate reconfigurable NoC components. The PowerPC executed software binary code that had overall control over the entire simulation. Their experiments suggested that the simulation could be finished within several seconds as compared to a couple of hours by a general software simulator. However, fast as this design was, it was far from an optimal solution. The size of the network, a 3×2 mesh, was relatively small compared with the ones in other works. This was resulted from the fact that most of the control logic was implemented with the hardware logic resources, which could have been implemented with software codes with better performance-to-utilization ratio. Also, as the hardware resources for source queues were uniformly and statically distributed among the nodes, they might not be fully utilized while simulating a non-uniform traffic pattern. In NoCOP, proposed by Liu et al.[21], a similar combination of hardware and software frameworks was also employed. Instead of having an embedded controlling processor, a hosting computer was introduced to coordinate the emulation process by communicating with FPGA device through USB connections. All other crucial NoC components, including the network, the traffic generators and receptors, were implemented with hardware resources.

Some other researchers focused on the potential of simulating a large scaled NoC instead of extreme simulation speed demonstrated in the above two works. In the work by Wolkotte et al.[22] which was one of the most representative works, a technique called Time-Division Multiplexing was employed to overcome the limitation of network size. Instead of implementing every node in a large size NoC and easily

hitting ceil of the hardware resources, they only physically implemented one node in the device and different nodes in the NoC were simulated sequentially on the only physical implemented node, one node per clock cycle. After each node's emulation was complete, its states would be stored to a large memory for use in the further simulations. Besides the FPGA chip housing the only one physical node, an SoC board housing two ARM processors and on-chip memory modules was connected to the FPGA board. The ARM cores were well coordinated with the FPGA chip so that the entire system could track the flow of data correctly. In this scenario, the communications between chips and memory accesses added huge overhead to the simulation speed. Besides, it is not power efficient to repeatedly move the states of the node back and forth between the FPGA board and SoC board. Also, implementation of one single node in the FPGA chip was inefficient as most the hardware resources were not utilized. The multi-board scheme also increased design cost and complexity. That said, the simulator demonstrated limited speedup over a pure System C simulator. The DART[23] also applied similar idea to the design, where a single I/O channel was implemented in FPGA rather than an entire node. Although the detail level and simulation accuracy were better guaranteed, the hardware resource utilization did not improve, and the simulation speed became worse.

AcENoCs[9, 10, 11] sought for a fast and cycle accurate simulator solution that also supported large simulated network size by combining and looking for tradeoff points between SW/HW frameworks. While hardware implemented TGs were fast, they consumed a huge amount of logic resources and constrained the available network sizes. In AcENoCs, the TGs were implemented in SW executed by on-chip MicroBlaze processors. Slower as the software TGs were, a great amount of hardware resources were released, making it possible to implement a larger interconnection network. The synthesis results suggested that AcENoCs was able to run simulations for network

up to 5×5 in size. On the other hand, the large simulated network size did not harm simulation speed too much. For the simulation of a 5×5 network, AcENoCs demonstrated 14-47X speed up over software simulators

CHAPTER III

SPAcENoCs DESIGN AND IMPLEMENTATION

In this chapter, we will briefly review the design AcENoCs, the basis that SPAcENoCs is implemented on. Also, we will discuss in detail the efforts beyond the work of AcENoCs that enables SPAcENoCs as a simulator for a larger scaled on-chip interconnection network compared with the applicable largest size presented in AcENoCs for the sake of further exploration of larger design space. To support a larger network without exceeding the hardware resource constraints, we re-introduce the time division multiplexing techniques utilized in the work of DART[23] while presenting better tradeoff between emulation speed and largest applicable network size. Under such a scheme, a large NoC, usually larger than 5x5, is divided into several, say N, slices, each of which is suitable for simulation on a single FPGA chip. The philosophy of the time division multiplexing lies on reuse of the hardware logic for different slices of the entire NoC. As it takes N cycles, one cycle for each slice, to emulate the whole network, the time division multiplexing method trades overall simulation speed for a larger applicable network size. This chapter will also discuss the design decisions surrounding the key hardware resources that impose constraints on the implementation of the simulations.

A. Review of AcENoCs Emulation Platform

As discussed in Chapter II, AcENoCs employed co-design of software[9] and hardware[10] frameworks for the sake of implementing an NoC simulator that supports large scaled network. The decision of SW/HW design was based on the observation and fact that the TG logic consumed a large portion of the hardware resources. It is, however,

much easier to implement by running software code, slower than the hardware implementation, on a light-weight processor that consumes less hardware budgets. In other words, the co-designed frameworks brought a tradeoff between the simulation speed and simulation size.

1. Platform Constitution

AcENoCs was implemented on Xilinx XUPV5 FPGA board[24] housing a Virtex-5 VLX110T FPGA device. A baseline design of AcENoCs contained several major components:

1. MicroBlaze Processor The MicroBlaze processor, running at 125 MHz, is a customizable processor supported by ISE toolkits. The MicroBlaze executes software code that generates traffic, injects and tracks flits, and collects ejected flits.

2. On-Chip Network: The on-chip network may accommodate up to 25 (in a 5×5 mesh) NoC routers, each of which is designed with pipeline and programmable delay. The interconnecting links between the nodes have constant one-cycle delay. Besides, a simplified Traffic Receptor (TR) is instantiated per Virtual Channel (VC) per router's output port to relief the buffering pressure in the software framework.

3. Peripheral Output Port: All runtime info is printed on host PC's display through the RS232 connection between an on-chip serial UART controller and the hosting PC.

4. Buses: AcENoCs utilized Processor Local Bus (PLB) to enable communications among other components within the design, including the processor, the network

and the UART controller. The software framework, the MicroBlaze processor, injects (ejects) flits by writing to (reading from) I/O registers specified by pre-defined physical addresses corresponding to the routers.

2. Emulation Flow

The entire simulation consists of iterations of simulation cycles and ends either all the generated flits have been traversed through the network as expected or the emulated cycles have reached the cycle limit, whichever comes earlier. The simulation will also be terminated abruptly if any unexpected error happens. Each simulation cycle contains several sequential phases:

1. One simulation cycle begins with asserting a rising edge on NoC's clock by writing one bit of "1" into NoC's designated clock register. This rising clock edge effectively activates NoC which receives injected flits, ejects flits, and signals packets' completion. The clock is deasserted several processor cycles later.

2. Traffic generation for each node/router is triggered and generated flits are put in a dynamic software source queue for injection. A flit is injected at one node if its source queue is non-empty and its FIFO buffer is not full.

3. The processor accesses NoC's status register to detect whether any packet's completion or error event has occurred at any node. The completed packets' info, as indicated by the status register, will be read out of the TR registers and sent to corresponding software TR.

4. The software TR deciphers received completed packet and merges the packet's delay with the NoC's aggregate latency data.

5. The NoC's clock is asserted again, initiating a new simulation cycle.

B. Time Division Multiplexing (TDM) Method

In this paper, the time division multiplexing method borrows the exact idea applied in many fields, including digital wireless communication and multi-task processing over one physical processor, the idea that multiple tasks, may share one or a few limited resources by allowing each of them exclusively owning the resource in different time slots. As long as the time slots are divided equally, the tasks appear to share the resource equally and simultaneously although they physically grab and utilize the resource in turn.

In our scenario, the task corresponds to the slices divided from the entire network; and the "resource" corresponds to the FPGA resources instantiated for any one of the slices. However, the implementation of TDM for this work is not as easy as TDM in digital wireless modulation where the bitstreams are usually independent from each other and no synchronization is required between any of these bitstreams. In SPAcENoCs, one slice communicates with its neighboring slices by both sending and receiving controlling and data messages, and the sequence of handling inter-slice data flits is key to the functionality of sequentially simulated slices. As SPAcENoCs is based on the work of AcENoCs and inherits combined framework of software and hardware implementation, the upgrade for TDM method requires coordination from both frameworks.

1. Hardware Framework Implementation

In SPAcENoCs, only a small portion of the entire network, a slice, is physically implemented in the FPGA chip. To enable data exchanges between logically adjacent slices which are simulated on the same physical slice, the slice is connected like a torus topology even for the simulation of a mesh network. A brief diagram of the slice's

torus-like connection is shown in Figure 1. The remainder of this subsection will discuss how the data are coordinated in a TDM scheme with the given physical instantiation.

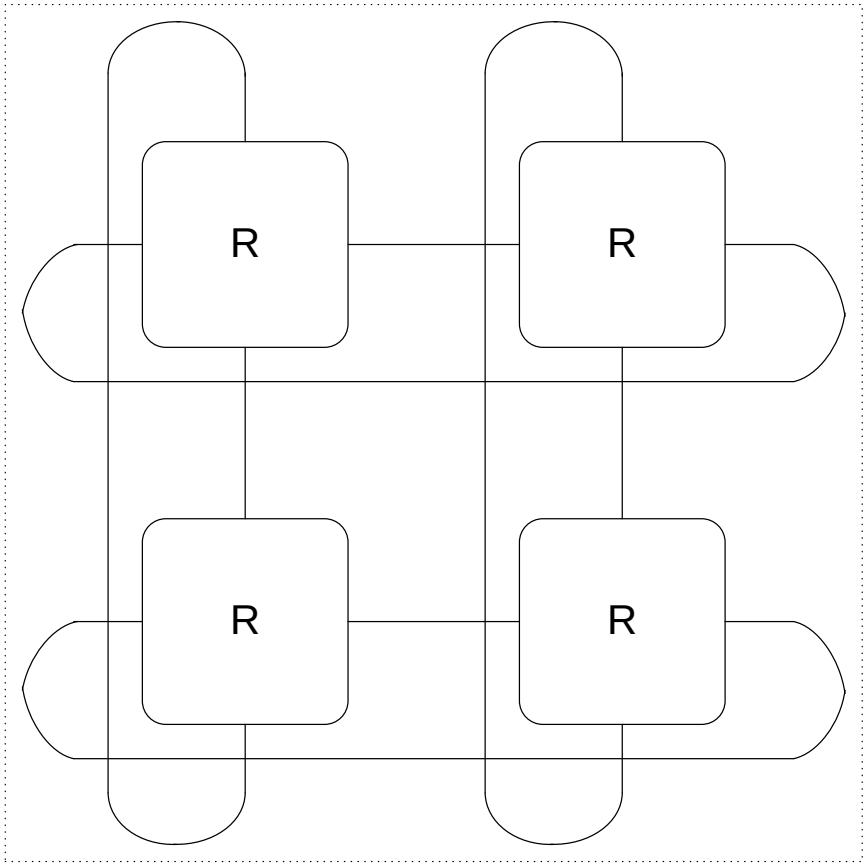

Fig. 1. An Example of Connections for a 2×2 Slice

In order to improve digital circuits' performance, throughput in other words, pipeline is very popular in practice. Under a pipeline scenario, a long delay logic path is divided into two or more shorter parts which may function in parallel over the inputs in different cycles instead of being kept waiting for the signal to propagate to the end of a long path in the non-pipelined scheme. The NoC routers, as described

in Chapter III, are designed with pipelines for the very sake of optimizing the overall throughput of the network. There are usually two parts of logic in a classic non-TDM pipelined design, which is also the case especially for a synchronous FPGA design:

1. Combinational logic: The combinational logic performs functional logic that is expected in current stage. Its inputs usually originate from the outputs of flip-flops (FFs) in the previous pipeline stage, and its output is stored into FF, which is introduced below, of current stage by the time when next clock edge, rising or falling depending on devices' implementation, arrives.

2. Flip-flop (FF, a.k.a Register): FFs are widely used in digital circuits to ensure synchronization of data across the entire design. They act like gates, or barriers, for data flow to be aligned to the clock regardless when the data are actually ready in different logic paths. In common practice, the synchronization point of an FF is the rising edge of the clock signal. When one clock's rising edge arrives, the FF will allow the data on its input port to pass through and to be presented on its output port, and it's output will not change, regardless of its input's transition, until next rising edge of the clock.

While it is pretty straightforward to understand how data are propagated through a series of pipeline stages under a non-TDM scheme, more attention and efforts are required to guarantee the data flows correctly in a TDM scenario when the hardware work discontinuously in terms of clock cycles for a single slice. The major efforts for hardware implementation to support TDM involve two parts: the intra- and inter-slice data exchange.

a. Intra-slice Data Exchange

The intra-slice data exchange, the data exchange within one slice, involves data flows that are pipe staged within routers in the same slice. When the time slot for one slice becomes active, the combinational logic in a certain stage should obtain its input from the result that its previous stage created in the slice's last ACTIVE cycle. If the combinational logic directly accesses the output of previous stage's FF as in a non-TDM scheme, it will find itself operating on something that is the result that ANOTHER slice, which just functioned in the previous physical cycle, has generated. On the other hand, when a slice's active cycle is coming to its end, preparing to hand the ownership of the hardware resource to anther cycle, we need to make sure all calculated results are properly staged without any risk of being overwritten by the ones produced by other slices. A single FF will not achieve this for sure as it can only stage, or store, the result for one cycle before it is overwritten whereas we need data to be safe for N cycles in an N-slice TDM scheme. Instead, we need a buffer.

The buffer should be such implemented that supports one read for providing input for next stage and one write for staging current stage's results. A FIFO is adequate but may be an over-kill as the rate of producer and consumer is constantly equal and detectors for over- and under-flow, which involves wasteful comparators, are unnecessary. In SPAcENoCs, the TDM buffer consists of one 0 to $N-1$ counter with encoded output, one $\log N$ to N decoder, one N to 1 MUX, and a file of flip-flops with N entries. The counter acts like the index for both read and write pointers. The output of the counter is also fed into decoder, the outputs of which are connected to the enable ports of the N-entry FFs to indicate which slot is supposed to be written at next clock edge. The counter also determines which entry should be read out of FF for the combinational logic. After several months of working in Marvell, the

author realized that there might be an alternative solution, in which a cyclic chain of registers replace the counter, providing shifted N-bit onehot vector as the output. In this solution, the onehot vector is shifted once every physical cycle, and enables one and only one data buffer's entry to accept producer's data on the write side. On the read side, this one hot vector drives the "selection" input of an AND-OR MUX that also acquires data from all FFs and provide only one set of data indicated by the one hot vector. The second solution may spare more Look-Up Table (LUT) blocks at the cost of several more flops.

b. Inter-slice Data Exchange

Under the inter-slice scheme, where data flow across the boundary between the slices, similar buffers are involved for properly handling and storing these data. However, the read and write pointers of the buffer behave differently compared with the ones under intra-slice scenario. In the intra-slice scheme, the output of the counter, acting as an ordering indicator of the active slice, may be used directly for indexing the buffer slots for read and write as the slice always operates on the data labeled with the same index. On the contrary, under inter-slice scenario, counter cannot be directly applied for the index of boundary-crossing data as their sources' and destinations' indices are different from the index of the active slice. A mapping algorithm, involving index and position of the active slice's and direction of the I/O port, is needed to calculate the both pointers.

Moreover, attentions are needed to avoid data alias for storage of data crossing the slices' boundaries which does not happen in an intra-slice scheme. Consider, in a simple case, two slice instantiated in a row and the inter-slice buffer between one pair of the routers that communicates over the slice boundary. After Router A, the one on the left, finishes its simulation at Cycle 1, and puts one flit in its east bound

output port; and then the active slot of Router B, the one on the right, has come for its simulation at Cycle 1. Router B should not take the output of Router A's output which was just generated in the previous physical cycle as that flit is supposed to be consumed by Router B at Cycle 2. Instead, the data, if any, Router B is supposed to consume are the ones produced by Router A at Cycle 0. Thus, at the point of when Router A prepares to inject data into the buffer at Cycle 1, we need to make sure that the flit, generated in Cycle 0 and yet consumed by Router B at Cycle 1, is NOT overwritten. In other words, any router on the edge of one slice should preserve its un-consumed stale data while producing new, or current, data.

That said, it is the simplest yet least efficient to achieve this by doubling the number of slots for a boundary-crossing buffer. This method simplifies the design of counter and counter-to-pointer mapping algorithm at the cost of more FFs. As discussed in the following section, budget for FF is one of the constraining resources and thus we should optimize the usage of FF, especially when these FFs are used for staging a large chunk of bits, more than 32 for each entry in our scenario, to realize a slice as large as possible. A closer observation that one slice consumes the same amount of data, in terms of number of buffer entries, as the ones it produces, may help us optimizing the buffer's utilization by re-use the entries whose data are just consumed. Thus, only one more redundant entry is needed to avoid data aliases. However, this solution complicates the design of mapping algorithm which varies greatly depending on slice's size, layout of the slices and directions of the I/O.

In SPAcNoCs, a tradeoff is made between resource utilization and design complexity by adding one additional entry for each row or column depending on the communication direction in which an inter-slice buffer is used in. For one NoC that is divided into $R \times C$ slices, the number of entries for horizontal, or east-west bound, inter-slice buffers is $R \times (C+1)$, and $(R+1) \times C$ for vertical communications, respec-

tively. One 0-to-$R-1$ row counter and one 0-to-$C-1$ column counter, instead of a single 0-to-$N-1$ counter, are employed to help calculating read and write pointers.

2. Software Framework Implementation

Compared with upgrade efforts for hardware implementation, software implementation is more trivial as the C code usually ensures better flexibility for re-organization. The upgrade efforts involve several major changes for the purposes of either maintenance flexibility or support of further functionality:

1. Establishment for TDM: In AcENoCs, one physical cycle, the cycle that executed on the physical NoC, is equal to one logical/virtual cycle, the cycle in which all parts of the NoC finishes one physical cycle's simulation. However, this is not the case for SPAcNoCs where one logical cycle consists of N, or $R{\times}C$ physical cycles for a sliced interconnection network. The software needs to address this issue by adding one more level of iteration outside the process of clock toggle, flits' injection and ejection. All the source and destination coordinates should also take the active slice's position into account and be adjusted accordingly.

2. Flit Format's Re-organization: In AcENoCs' flit format, the position of source or destination node is represented by a pair of X and Y coordinates where both X and Y are presented and transmitted with 3 bits. This representation effectively confines the largest applicable size to an $8{\times}8$ network regardless of the availability of the hardware resources. In SPAcENoCs, the bits representing X and Y are extended with one bit, enabling the simulation of an interconnection network as large as 16×16 which is larger than most, if not all, of state-of-art NoC implementations. The extension of the coordinates results in shrinking of

the "PacketID" from 12 bits to 10 bits, adding the risk of packet aliasing. We can resolve this by assigning source-node-specific packet ID rather than a global one. The tracking infrastructure also needs corresponding updates.

3. Removal of Unrolled Loops: AcENoCs' software implementation employed a technique called "Loop Unrolling" commonly known for reducing branch mis-predictions penalties and thus improving the performance of the processor. However, the manually unrolled loops impede the efforts for parameterizable and reconfigurable implementation which is one of the crucial features of SPAcENoCs. Thus, the manually unrolled loops are removed in our software framework to enable quick deployment and to avoid generating multiple copies of source codes to support various slice and network sizes. The loss of performance due to removing unrolled loops is marginal as the overhead for determination on loop condition is negligible compared with the code size of the body of the iteration. Also we may rely on compiler which unrolls the loops as part of its optimization, which does not impose and difficulty for reconfigurable code writing.

4. Code Re-organization: In a normal framework of C codes, source codes contain two parts, the header (.h) file containing macros and definitions of customized data structures, and C (.c) file containing actual data structures and functional codes. In normal practice, declaration of global variables should be avoided in header files and a header file should be protected by "ifdef" macros to prevent multiple instantiation when it is included by multiple other files. However, AcENoCs framework did not follow these principles in general, which caused several issues during the attempt for upgrading it into SPAcENoCs. These issues are resolved by separating declaration and definition of global variables into header and C files, and by replacing inclusion of a C file with its corresponding

header file.

C. Design Over Resources

In SPAcENoCs framework, hardware resources have always been the major focus since the beginning of the work. As multiple design decisions interact upon the usage of the resources, it is difficult to predict the optimal size for both the slices and the entire network, even given the work of AcENoCs. Several configurations varying from design parameters of routers, slices and the whole network, implementation input FIFOs and even the software setup may affect the applicable design options. As most of the combinational logic can be reused and does not scale while applied for TDM implementation, the budget of LUT, the hardware to implement combinational logic, appears not to be our major concern over resource availability. However, the MUX does scale proportional to the number of slices. The budget on storage sources is also imposing a restrictive over the design of SPAcENoCs. This section will discuss implication of several major design decisions and their implications over these available resource budgets.

As SPAcENoCs is developed on the basis of the work of AcENoCs, the configuration of AcENoCs is a good reference as the starting point of exploring design feasibility. The major parameters of AcENoCs' implementation that impact the resources utilization are listed in Table I

1. LUT Resources

According to AcENoCs, LUT resource is the bottleneck for the potential of realizing a larger interconnection network. The method of TDM also aims at reducing the

Table I. Summary of Resource-Related Parameters

Feature	**Parameter Value**
Network Size	2D Mesh (2x2, 3x3, 4x4, 5x5)
Input Ports per Router	5 (4 from neighbors and 1 from local)
Virtual Channels per Input Port	2
Depth per Virtual Channel	8
Link Width/Flit Size	32 bits
Max Programmable Delay Cycles	3
BRAMs Used by CPU	64

usage of combinational logic which consumes LUT resources. However, the size of multiplexers (MUXes) will scale as the number of slices increments because they are largely needed for selecting desired data out of N entries in the data buffers described in the previous sections.

Actually, as we will see by discussing other limiting factors, LUT resource is still the major restriction over the attempt for implementing more and larger slices in SPAcENoCs. In theory, the ultimate applicable entire network size is determined by the size of each slice and total number of the slices. However, while it is easy to enlarge the slice's size, 5×5 as in AcENoCs, it is difficult to achieve a reasonable number of slices for a 5×5 slice because of scaled requirements for LUT to implement MUXes. To make it worst, a larger slice will demand more routing resources, consume more LUTs and make the budget even tighter. In other words, the number and the size of slices are fighting each other under given resources for us to achieve an optimal emulatable network size. To realize the optimal/largest network size, a tradeoff between the two factors are needed by shrinking the slice's size and utilizing released resources to scale the number of slices. We have attempted several slice setups and the optimal overall

size we could get is a 9×6 network divided into six 3×3 slices, more than twice as large as a 5×5 network, the largest applicable size that AcENoCs could achieve. As the LUT resource is the key constraint of SPAcENoCs, the following discussion will base on the settled configuration of the slices.

2. Block RAM V.S. Distributed RAM

In Xilinx XC5VLX110T FPGA[24] chip, 148 physical RAM blocks (BRAMs) dedicated for RAM-based IP applications, are instantiated. Each of these BRAMs, with 36K-bit capacity, can be instantiated memory that supports one read port plus one write port whose clocks may be either synchronous or asynchronous to each other. On the other hand, XC5VLX110T contains Distributed Memory modules, up-to 1120 Kb in size, that may also be used for RAM-like modules.

Based on the AcENoCs' settings listed in Table I, the overall demand for register/data storage is 3475 bits per router, the major amount, more than 3100 bits of which is consumed by input buffers and programmable delay registers. Then in the 9×6 network, the total register consumption would be around 188K bits, far away from the limitation of distributed memories or BRAMs. Virtual Channel (VC) is the major part of the input unit and largest consumer of registers/FFs throughout the entire router. Under a design of Each VC can be implemented with one BRAM or distributed memories using the macros that Xilinx ISE design toolkit[25] provided:

1. BRAM Implementation: For 6-slice network implementation, the total storage usage for a single VC is $32(bits\ per\ entry)\times 8(entries\ per\ slice)\times 6(slices) = 1536\ bits$ which may fit into one BRAM. As SPAcENoCs inherits the major framework of AcENoCs, 64 of these BRAMs (256KB in size) are allocated for the use of MicroBlaze IP processor, and 84 BRAMs are available for implementing

VCs. In a router with 5 input ports with 2 VCs per port, 10 VCs or BRAMs are needed and 90 BRAMs are needed for a 3×3 slice, which exceeds the BRAM's quota. However, we can still implement VCs with BRAMs for 8 routers and "borrow" some distributed memory to implement the VCs for the last router.

2. Distributed Memory Implementation: This method may take advantage of the fact that the memory is such distributed may impose less routing constraints compared with BRAM implementation. Also, the actual test showed that distributed memory implementation has better timing behavior than the BRAM implemented ones.

3. Memory Usage

In implementation of the processor, BRAM are used as its RAM memory which will accommodate all the text segment, stack segment and data segment. The text segment contains the compiled binary code that does not vary much which the simulated size of the network scales. The stack segment is used for stack which will be occupied by local variables that are used in various functions. The data segment is used for holding global variables and queues that track the packets to inject and ones ejected from the network. As the global space occupied by these queues is proportional to the total number of the nodes, the overall memory space will also limit the overall size of emulated interconnection network. The data space required by the 9×6 network can be fit into the given memory space.

Application of off-chip DRAMs is also an alternative solution which may enable software framework tracking substantially more queue for larger network. However, this will degrade the simulator's performance by adding latency of accessing off chip memory. The support for DRAM will require more logic resources for Memory Man-

agement Unit (MMU), imposing a tighter constraint over emulation size. Also, as discussed above, memory space is not the most restrictive factor over the emulatable size. All said, we don't apply DRAM memories in the implementation of SPAcENoCs.

CHAPTER IV

SPAcENoCs DESIGN VERIFICATION AND PERFORMANCE ESTIMATION

In this chapter, we will demonstrate how the challenge of verifying design of SPAcENoCs is resolved by adding reference models and verify the both framework separately. In our solution, software framework is verified first and becomes part of reference model and stimulus generator for the hardware framework verification that follows. We will also conclude this chapter by a discussion over the performance of the SPAcENoCs based on the results of AcENoCs.

A. SPAcENoCs Desgin Verification

Design verification has always been a vital process during the procedure of project and product development, and usually requires more human resources and takes longer time than the design work. Verification of FPGA design is a constant challenge among the verification tasks due to its nature of low internal observability as well as the time spent in repeatedly compile-program iterations for fixes to be patched into erroneous design. This challenge further deteriorates in a design, like SPAcENoCs, where software and hardware frameworks are combined and intensively interact with each other and thus it is difficult to verify the two frameworks individually.

However, difficult as the verification of a combined design is, it is still possible to separate the two frameworks and verify either of them individually by designing an intermediate model in place of the actual framework. This intermediate model, or reference model, should be easy to implement, bug free, and cooperative based on the interface and protocol given by either side of the framework.

1. Software Framework Verification

As discussed in previous Chapter III, the software framework behave as a data generator and collector, and the simulation task, data processing task in other words, is handed over to the hardware framework by injecting flits in to the network and receiving them after the flits traversed their designated path inside the NoC. Thus, the software framework would not function correctly if the NoC, its hardware counterpart, were removed for the sake of individual verification. We then need a simple and correct model, called Virtual NoC, plugged into the software framework so that it makes software framework believe it is interacting with a physical NoC without the presence of the physical network.

a. Virtual NoC Implementation

The Virtual NoC is written in C++ which is easy to implement by employing Standard Template Library(STL) when dynamic queues are involved in this reference model. For verification and testing purposes, it should at least support following features:

1. Flit Comprehension: The ability to comprehend the flits, especially the head flits, is in the first place of the features to enable correct handling of the flits. Whenever a head flit is injected, the coordinates of source and destination nodes are extracted and parsed. An instant check is conducted to make sure the flit is injected correctly into expected source node.

2. Flit Queue Management: After a flit is injected, it is put in an internal tracking structure. Instead of simulating how it is traversed through designated path inside the network, which is worth a light-weight software NoC simulator and a new individual paper, we simply put it in queue-like data structure. Flits

in the in-flight queues are tracked using time out mechanism in which a flit's life time, set as an arbitrary but reasonable positive number when injected, decreases to zero over time before it is ejected and returned to the software data collector. This simplification, without violating NoC's behavior model and software-hardware protocol, reduces verification complexity, and avoids unnecessary debugging efforts for the Virtual NoC itself.

3. Time-Out Mechanism: As discussed above, a flit's life time decrements as time elapses. However, the flits should be such carefully handled that their life time decrements only when their active cycle has just finished (or arrived). For simplification, each flit's active cycle is the same as the one of the slice that it is injected. In other words, a flit's life time is reduced once every N cycles in an N-sliced network. When a flit's life time reaches zero, it is put into another Traffic Receptor(TR) queue of its destination node. As the actual value of a flit's or a packet's lifetime does not affect the functionality of the software framework, for the sake of simplification and reducing chances of bugs, the Virtual NoC supports only constant initial life time that is set during a flit's injection. Also, the constant life time eases the management of in-flight queue by enqueueing newly injected flits at the tail of the queue and all the flits are naturally sorted by their life time. At the end of every active cycle when we need to eject time-out flits, we just need to dequeue the head of the queue until the flit in the head is not timed-out or the queue is empty.

4. Virtual Traffic Receptor: The virtual TR is a software model and replacement of the physical TR instantiated as part of the physical NoC. It collects ejected flits, checks integrity of the packet and reports the completion of a packet if all its flits are received in order or an error message otherwise. Each nodes' virtual

TR will check the flits in its TR queue every active cycle and send message back to the software framework for further bookkeeping. The virtual TR will trigger an error message and abort the simulation if flits are received out of order (because software framework injected out of order by mistake) or ejected at wrong destination node (most likely by bugs inside the Virtual NoC). Upon a report of a packet's completion, a report flit, in the same format of a physical TR's output, is wrapped and sent back through the I/O between Virtual NoC and software framework.

5. Global Slice Counter: The global slice counter corresponds to the counter in physical NoC that indicates which slice's active cycle has arrived.

b. Virtual NoC Connection

Given that Virtual NoC is completed, another question arises that how we should connect, or attach, the Virtual NoC module to the software framework without too many modifications in the software framework. This can be resolved by overriding the system functions that the software uses to communicate with physical network.

In the software code, the controlling logic injects flits or toggles physical network's clock by using system macro called "FPGA_TOP_mWriteReg" provided by Xilinx. Similarly, it collects status, ejected packet info and any possible errors returned from hardware framework by calling "FPGA_TOP_mReadReg". For the these two functions, two of the function arguments are both needed: the base address, the offset, and the read/write physical address is calculated by $base\_addr + offset$. And one more arguement is need for the write function to indicate the value to be written into the designated physical address . As each node of the physical NoC has two unique physical address, one for accepting writes and one for providing read data from/to

the software side respectively, we may tell from the physical address that which node is intended to be written/read.

That said, we can connect the Virtual NoC to the software side by replacing the content of the I/O functions while still applying the identical function signatures. To achieve this, a new set of I/O functions, which exchange data between the Virtual NoC and software side, is written as part of Virtual NoC. Also, the "fpga_top.h" header file, which provides links to the mentioned library functions, is replaced by a new header file pointing to the new functions. In our practice, in order to compile code for either verification or actual simulation purposes, the replacement of header files as well as the code for Virtual NoC are protected by "ifdef/endif" pairs so that we can easily compile the code for different purposed by applying just one different compile option to the command line.

All said, the entire verification work over the software side is done on PC platform using GCC compiler so that we do not have to verify the software side on the actual FPGA platform which is slower and less convenient.

2. Hardware Framework Implementation

Amongst various challenges of verifying a hardware residing in FPGA, re-programming time and low observability are the two major ones that prevent developers and researchers of FPGA based projects. Simulation tools running on PCs are then introduced to provide better observability without physical access to the FPGA devices. In the verification practice of SPAcENoCs, off-line simulations are employed to enable swift and convenient debugging.

That said, the generation of stimulus has become an issue especially when we need a relatively large test set to obtain enough confidence over the design of hardware framework. It is too inefficient, if not impossible, to generate these stimulus

manually. The software framework, which has been verified by introducing Virtual NoC, can now be trusted and deployed for generating stimulus in the syntax of Verilog HDL. To achieve this goal, the same "overriding" technique is applied on write functions while data collecting part is disabled as only the injection functionality is needed for the stimulus generation. Whenever the software calls the overridden write function, either trying to toggle clock or to inject a flit, it prints several lines of stimulus following Verilog HDL syntax into an output file. For example, when the software tries to toggle the NoC's clock by writing to the physical address specified for the clock, two lines of Verilog code containing clock signal value as well as a fixed simulation time delay will be printed as output. Analogously, Verilog statements assigning flit data to NoC's data input and asserting the flit valid signal will be printed when the software tries to inject one flit into a specific node. This output file is later concatenated to other Verilog language fragments, which contains essential connectivity, test environment setups, and instantiations of physical components, to form a complete and legal Verilog one to be used for off-line simulation. This method is called "Generic Programming" in the field of pure software development.

The checking of the simulation result is then visually checked by examining both waveforms and output logs. While it is challenging by doing so, the physical TRs embedded as part of the NoC free us from most of the checking efforts. We may, however, apply software frameworks for off-line checking, which more efficient in verification but complex to deploy. Thus, this method is not extensively discussed in our work.

3. Verification Flow Automation

Running automated scripts is always convenient and error-proof compared with a sequence of manual operations in Graphic User Interface(GUI), especially under the

scenario where the sequence of operations are repeated many times making the human operator bored and prone to errors. Also it suffices to root most of the design flaws by reviewing logs printed out by FPGA simulators. Moreover, the non-GUI environment is favored by lovers of Working-From-Home (WFH), by providing quicker response while requiring less network bandwidth without compromising connection security compared with a GUI remote desktop. In the practice of developing and verifying SPAcENoCs, the automated scripts used by the GUI environment are studied, extracted and hacked (without violating the toolkits' copyright, though) to run under shell command prompt which works perfectly either locally or remotely through an SSH connection.

B. SPAcENoCs Evaluation

In this section we will discuss the hardware resource utilization of SPAcENoCs and estimate the speed-up over OCIN_TSIM based on the actual result of AcENoCs.

1. Resource Utilization

As mentioned in Chapter III, the largest applicable network size is 9×6 with six slices, each of which contains $3 \times 3 = 9$ baseline routers presented in AcENoCs. The ultimate utilization rate of FPGA logic slices (which contains LUT and FFs) is 99%, almost hitting the ceiling of the resource budget. Compared with the 3×3 NoC implemented in AcENoCs which used only 21% of the LUTs, we may tell that most of logic resources are used for the implementation of MUXes and long boundary-to-boundary routes.

Several adjustments in either configuration or implementation may help with the utilization rates:

1. Application of BRAMs for input FIFOs may reduce the usage for LUT, FFs and distributed memories.

2. A shallower FIFO will effectively reduce the cost of logic slices.

3. A more aggressive, and thus slower, floor planning and routing algorithm may help with the final implementation as well.

4. SPAcENoCs has not attempted options with slice smaller than 3×3 due to time pressure. However, it is possible to improve overall network size by implementing smaller sized slices which may release more resources from routing to enable more number of slices to manage a larger overall network size.

2. Performance Estimation

As SPAcENoCs applies the TDM methods known for trading speed/time for space, one question may arise that whether it is still able to gain reasonable speed-up over OCIN_TSIM compared with the numbers claimed by AcENoCs. Even without the actual simulation on either side, we may, however, by applying very simple mathematical calculations over the results of AcENoCs, estimate the ultimate speed-up of SPAcENoCs over OCIN_TSIM. In the work of AcENoCs, the FPGA aided simulation accelerator, running at 125 MHz, demonstrated 14-47X speed-up, depending on flit injection rate, over OCIN_TSIM running on PC platform equipped with Intel Xeon processor running at 3.2 GHz for a 5×5 NoC simulating synthesized workload. Also, on AcENoCs, simulation of a 3×3 NoC is generally 6 times as fast as simulation of a 5×5 network. Before starting our estimation, three more assumptions are needed:

1. The speed of software simulations run on PC platform scales with the total number of NoC routers, given a fixed injection rate.

2. The overall time spent on simulation on AcENoCs or SPAcENoces is the time spent by software framework, because the simulation on hardware framework is parallel to that on the software framework and the time the hardware spent can be shadowed by the software side.

3. The overall time spent by SPAcENoCs for an N-sliced NoC is roughly N times of the time cost for AcENoCs to simulate a NoC with the same size as the slice, because one logical cycle in SPAcENoCs roughly equals to aggregation of$N$ physical cycles, the latter of which spends the same time as AcENoCs.

Let $T_{PC,M\times N}$ denote the time spent on PC-based simulation of an $M\times N$ NoC, and $T_{FPGA,M\times N}$ for the time spent on SPAcENoCs for the same sized NoC. Then we may have:

$$\begin{aligned} SPEEDUP_{9\times 6} &= \frac{T_{PC,9\times 6}}{T_{FPGA,9\times 6}} \\ &= \frac{T_{PC,3\times 3}\times 6}{T_{FPGA,3\times 3}\times 6} \\ &= \frac{T_{PC,3\times 3}}{T_{FPGA,3\times 3}} \\ &= \frac{T_{PC,5\times 5}\times \frac{9}{25}}{T_{FPGA,5\times 5}\times \frac{1}{6}} \\ &= \frac{T_{PC,5\times 5}}{T_{FPGA,5\times 5}} \times \frac{54}{25} \\ &\approx 2 \times \frac{T_{PC,5\times 5}}{T_{FPGA,5\times 5}} \end{aligned}$$

This effectively means the SPAcENoCs is estimated to gain around 28–94X speedup, based on the results of AcENoCs, over OCIN_TSIM on a simulation of 9×6 NoC. Several ways may further improve the performance of SPAcENoCs:

1. Higher clock frequency, although not applicable by ISE, may further accelerate speed of the MicroBlaze Processor and thus reduce the overall simulation time

2. More powerful main processor that has better Instruction Per Cycle (IPC) will reduce the simulation time

3. Further handcrafted source code and aggressive compiler optimization may help reduce the emulation time as well.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

A. Conclusions

While the VLSI technology rapidly advancing and enabling increasing number of processors to be integrated on a single silicon chip, the traditional communication on the basic of shared bus has become the impedance for the multi-processors to fully exploit their performance as the nature of shared bus does not favor the communication where multiple parties fight to acquire the privilege for exclusively ownership of the bus. NoC, or the interconnection network has come into the play as it enables high throughput low latency and robust data communication amongst the multiple cores.

Due to the stiff time-to-market pressure for development of new products, an NoC simulator is always desirable to be fast, cycle accurate and reconfigurable to aid exploration of vast design space contributed by several major design aspects with various flavors of options. It is generally difficult for the software based simulators, serialized in nature, to accomplish fast simulation without sacrificing simulation accuracy, FPGA devices are introduced to accelerate NoC simulation by utilizing its parallel and reconfigurable nature.

While most of FPGA aided NoC simulators managed to either accelerate simulation on a small sized NoC or emulate a large interconnection network, few of them were able to accelerate the simulation over a large scaled NoC. Based on the efforts of, and inheriting highlighted features of AcENoCs, which attempted to accelerate simulation over a large scale NoC without sacrificing simulation speed and accuracy, the SPAeENoCs further exploits such possibility for even larger NoCs by employing TDM techniques. As LUT resource is the major restrictive constraint to support large

simulated network size, the major part of combinational logic which consumes precious LUT resource, may be re-used under TDM scheme to enable FPGA to simulate different regions of the network in different time slots. As a result, the SPAcENoCs FPGA simulator, using the same resource that AcENoCs was allocated, is able to simulate a network as large as more than twice the size of largest applicable size of AcENoCs. With further refinery in code writing and implementation adjustment, it is believed larger network size is also possible over the same FPGA device by using the philosophy employed by SPAcENoCs.

During the verification of SPAcENoCs, we employed techniques like device virtualization and generic programming to overcome the verification and debugging challenges that arose in the design with combined software and hardware frameworks. And all these techniques can be applied and further improved to aid verification procedures in the future work. The estimated performance speedup of SPAcENoCs running at 150 MHz for a 9×6 NoC is 28–94X faster than OCIN_TSIM which is run on processors operating at 3.2 GHz, showing the possibility and feasibility of FPGA aided simulation over large scaled NoCs.

B. Future Work

While managing a new height in terms of finding the largest simulatable network size, SPAcENoCs is just a starting point that can be further developed and improved in following area or directions:

1. While SPAcENoCs only supports symmetric/identical sized slices, it is possible for it to support asymmetric slices. For example, by support asymmetric slices, we may simulate a 7×7 NoC by slicing it into four, or more if necessary, parts: 4×4, 4×3, 3×4, and 3×3. The asymmetric slicing adds more flexibility of

the overall network size.

2. Instead of using the on-chip MicroBlaze IP processor which is relatively weak in terms of both frequency and computing capability, it is possible to replace the core with a USB port that communicates to a hosting PC machine running software simulator like OCIN_TSIM. In such a scenario, the hardware resources can be fully applied for implementing simulated NoC; and the powerful PC processor takes the charge of control-oriented tasks like traffic generation, flits injection and ejection, and statistics collection. As a result, both sides may fully exploit their strength and further improve the performance of NoC simulation.

3. It is also possible to further explore possible network sizes by "playing" with the slices' layout and sizes. Although 9×6 is claimed as the largest applicable network size by this paper, there are still chances to implement an even larger network.

REFERENCES

[1] G. Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilitie," AFIPS Conference Proceedings, vol. 30, pp. 483–485, 1967.

[2] M. D. Hill and M. Marty, "Amdahls Law in the Multicore Era," *Computer*, vol. 41, pp. 33–38, July, 2008.

[3] A. Leon-Garcia and I. Widjaja, "*Communication Networks.*" New York, NY:McGraw-Hill Education, 2004.

[4] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks.* San Francisco, CA: Morgan Kaufmann Publishers, 2004.

[5] W. J. Dally and B. Towles, "Route packets, not wires : On-chip interconnection networks," in *Proc. Design Automation Conference*, Las Vegas, NV, Jun. 2001, pp. 684–689.

[6] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz mesh interconnect for a teraflops processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[7] Tilera Corporation, "Tile64 processor family," [Online]. Available: http://www.tilera.com/products/processors.php, Accessed Feb. 2009.

[8] P. Gratz, K. Sankaralingam, H. Hanson, P. Shivakumar, R. McDonald, S. Keckler, and D. Burger, "Implementation and evaluation of a dynamically routed processor operand network," in *Proc. 1st ACM/IEEE International Symposium on Networks-on-Chip*, Princeton, NJ, May 2007, pp. 7–17.

[9] V. Pai, "HW/SW Codesign and Design, Evaluation of Software Framework for AcENoCs: An FPGA-Accelerated NoC Emulation Platform," M.S. thesis, Texas A&M University, College Station, TX, 2010.

[10] S. S. Lotlikar, "Design, Implementation and Evaluation of A Configurable NoC for AcENoCs FPGA Accelerated Emulation Platform," M.S. thesis, Texas A&M University, Collete Station, TX, 2010.

[11] S. Lotlikar, V. Pai, and P. Gratz, "AcENoCs: A flexible HW/SW platform for FPGA accelerated NoC emulation," in *Review 24th International Conference on VLSI Design*, Chennai, India, Jan. 2011.

[12] T. Kogel, M. Doerper, A. Wieferink, R. Leupers, G. Ascheid, H. Meyr, and S. Goossens, "A modular simulation framework for architectural exploration of on-chip interconnection networks," in *Proc. 1st IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, Newport Beach, CA, Oct. 2003, pp. 7–12.

[13] Z. Lu, R. Thid, M. Millberg, E. Nilsson, and A. Jantsch, "NNSE: Nostrum network-on-chip simulation environment," in *Proc. Swedish System-on-Chip Conference*, Stockholm, Sweden, Apr. 2005, pp. 1–4.

[14] S. Pestana, E. Rijpkema, A. Radulescu, K. Goossens, and O. Gangwal, "Cost-performance trade-offs in networks on chip: A simulation-based approach," in *Proc. Design Automation and Test in Europe Conference*, Paris, France, Feb. 2004, pp. 764–769.

[15] D. A. Siguenza–Tortosa and J. Nurmi, "VHDL-based simulation environment for Proteo NoC," in *Proc. High-Level Design Validation and Test Workshop*, Cannes, France, Oct. 2002, pp. 1–6.

[16] S. Prabhu, B. Grot, P. Gratz, and J. Hu, "Ocin_tsim - DVFS aware simulator for NoCs," in *Proc. 1st Workshop on SoC Architecture, Accelerators and Workloads (SAW-1)*, Bangalore, India, Jan. 2010, pp. 1–8.

[17] K. Goossens, J. Dielissen, O. Gangwal, S. Pestana, A. Radulescu, and E. Rijpkema, "A design flow for application-specific networks on chip with guaranteed performance to accelerate SoC design and verification," in *Proc. Design, Automation and Test in Europe*, Munich, Germany, Mar. 2005, pp. 1182–1187.

[18] N. Genko, D. Atienza, G. De Micheli, J. Mendias, R. Hermida, and F. Catthoor, "A complete network-on-chip emulation framework," in *Proc. Design, Automation and Test in Europe*, Munich, Germany, Mar. 2005, pp. 246–251.

[19] N. Genko, D. Atienza, G. De Micheli, and L. Benini, "Feature - NoC emulation: A tool and design flow for MPSoC," *IEEE Circuits and Systems Magazine*, vol. 7, no. 4, pp. 42–51, 2007.

[20] A. Jalabert, S. Murali, L. Benini, and G. De Micheli, "xpipesCompiler: A tool for instantiating application specific networks on chip," in *Proc. Design, Automation and Test in Europe Conference*, Paris, France, Feb. 2004, pp. 884–889.

[21] P. Liu, C. Xiang, X. Wang, B. Xia, Y. Liu, W. Wang, and Q. Yao, "A NoC emulation/verification framework," in *Proc. Sixth International Conference on Information Technology: New Generations*, Las Vegas, NV, Apr. 2009, pp. 859–864.

[22] P.Wolkotte, P. Holzenspies, and G. Smit, "Fast, accurate and detailed NoC simulations," in *Proc. First International Symposium on Networks-on-Chip*, Princeton, NJ, May 2007, pp. 323–332.

[23] D. Wang, "An FPGA-based Accelerator Platform forNetwork-on-Chip Simulation," M.S. thesis, University of Toronto, Toronto, Canada, 2010.

[24] Xilinx Inc., "Xilinx university program XUPV5-LX110T development system," [Online]. Available: http://www.xilinx.com/univ/xupv5-lx110t.htm, Accessed Dec. 2009.

[25] Xilinx Inc., *Embedded System Tools Reference Manual* [Online]. Available: http://www.xilinx.com/support/documentation/sw_manuals/xilinx11/est_rm.pdf, Accessed Dec. 2009.